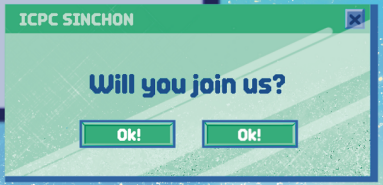
2024 ICPC Sinchon SUAPC Summer Solution

신촌지역 대학생 프로그래밍 동아리 연합

2024년 8월 25일



	문제	의도한 난이도	출제자
A	SWAPC	Easy	이지언ez_code
B	평균 구하기	Hard	박상수molamola
C	숫자 놀이	Medium	곽우석bubbler
D	축지법	Medium	전우제kiwiyou
E	n, 3n, 5n	Hard	곽우석bubbler
F	사탕 배달	Challenging	정치훈ANZ1217
G	지워진 ETT	Medium	김도훈dohoon
H	잊음을 논함	Challenging	노현제hjroh0315
I	시험 주행	Easy	김영현kipa00
J	a11y	Hard	박상수molamola
K	완전 이진 트리와 쿼리	Medium	정치훈ANZ1217
L	허술한 보안 프로그램	Medium	서종환mujigae
M	나무 키우기	Hard	박상수molamola

A. SWAPC

`string, implementation`

출제진 의도 – **Easy**

- 제출 70번, 정답 44팀 (정답률 62.857%)
- 처음 푼 팀: **PS:Endgame**[연세대학교], 1분
- 출제자: 이지언[ez_code]

A. SWAPC

- 간단한 문자열 구현 문제입니다.
- P와 C의 위치를 저장한 후, min(P의 개수, C의 개수)만큼 위치를 바꾸면 됩니다.
- 총 시간 복잡도는 $\mathcal{O}(N)$ 입니다.

B. 평균 구하기

`binary_search`

출제진 의도 – **Hard**

- 제출 24번, 정답 4팀 (정답률 16.667%)
- 처음 푼 팀: **Redshift**서강대학교, 15분
- 출제자: 박상수molamola

B. 평균 구하기

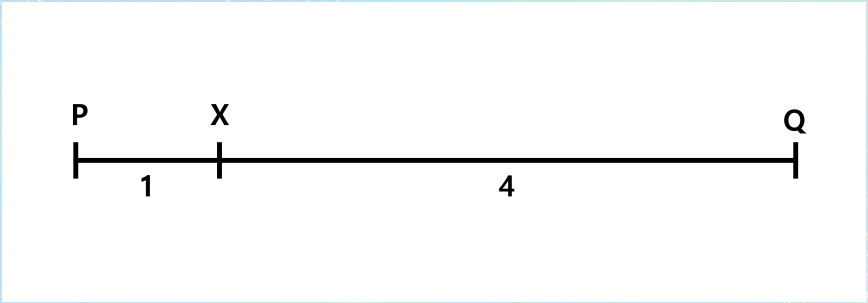
- N개의 점을 수직선 위에 위치시켜 봅시다.
- 할 수 있는 연산은 두 점을 골라 중점을 찍는 것입니다.
- 이 연산을 사용해서 전체 평균점을 구하면 됩니다.
- 다양한 풀이가 있는데, 그 중 귀납법을 사용한 풀이를 소개합니다.

B. 평균 구하기

- (처음 $n-1$개 점 평균)을 P, n번째 점을 Q라고 합시다.
- P와 Q의 $1:(n-1)$ 내분점을 구하면 이 점이 n개 점의 평균이 됩니다.

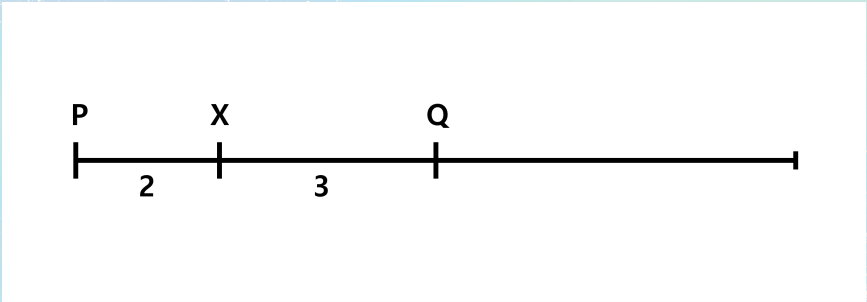
B. 평균 구하기

- P와 Q의 $1 : 4$ 내분점 X를 구해 봅시다.



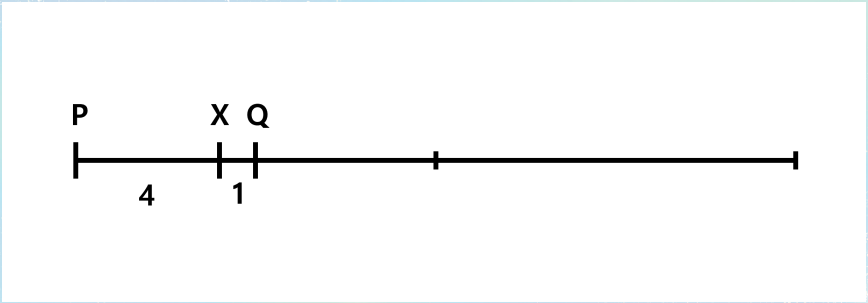
B. 평균 구하기

- P와 Q의 중점을 찍고, X를 포함한 쪽으로 구간을 좁힙니다.



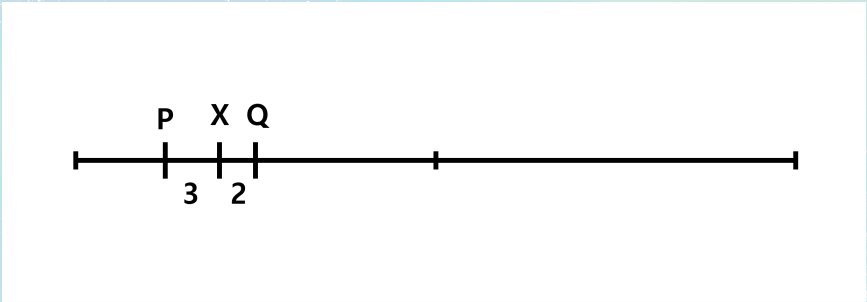
B. 평균 구하기

- P와 Q의 중점을 찍고, X를 포함한 쪽으로 구간을 좁힙니다.



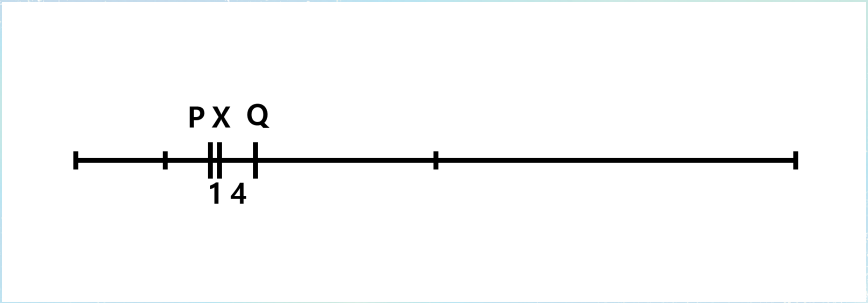
B. 평균 구하기

- P와 Q의 중점을 찍고, X를 포함한 쪽으로 구간을 좁힙니다.



B. 평균 구하기

- P와 Q의 중점을 찍고, X를 포함한 쪽으로 구간을 좁힙니다.



B. 평균 구하기

- 위 과정을 각 점에 대해 넉넉하게 100번 정도 수행하면 됩니다.
- 이 때 최종 오차가 $2^{-100} \times N \times 10^9$ 이하임을 보일 수 있습니다.

C. 숫자 놀이

constructive

출제진 의도 – **Medium**

- 제출 90번, 정답 33팀 (정답률 36.667%)
- 처음 푼 팀: **goraani never stop**[서강대학교], 12분
- 출제자: 곽우석[bubbler]

C. 숫자 놀이

- 먼저, 두 수를 어떻게 고르더라도 칠판에 쓰여 있는 값의 최댓값을 증가시킬 수 없음을 관찰합니다.
- 따라서, 처음에 칠판에 쓰여 있는 값들의 최댓값인 N을 마지막까지 그대로 두고 나머지를 가능한 한 작은 수 1개로 만드는 것이 유리할 것이라고 예상할 수 있습니다.
- 이제 $1, 2, \cdots, N-1$을 가지고 가능한 한 작은 수를 만들어 봅시다.

C. 숫자 놀이

- $a+b$와 $|a-b|$의 홀짝성은 항상 같습니다.
- 따라서 $1, 2, \cdots, N-1$을 가지고 만들 수 있는 수는 $1+2+\cdots+(N-1)$과 홀짝성이 같은 수들 뿐입니다.
- 그 중 가장 작은 수는 $(1+2+\cdots+(N-1)) \bmod 2$가 됩니다.
- 어떻게 하면 이 값을 만들 수 있을까요?

C. 숫자 놀이

- 이는 의외로 간단한 방법이 있습니다.
- $N-1$과 $N-2$를 골라서 1로 만들고, $N-3$과 $N-4$를 골라서 1로 만들고, $\cdots$를 반복하면 최종적으로 여러 개의 1이 남습니다.
- 그 다음에는 1을 두 개씩 짝지어서 0으로 만드는 것을 반복하면 1이 0개 또는 1개가 남고 나머지는 0이 됩니다.
- 칠판에 있는 0은 아무 수와 짝지어서 제거할 수 있습니다. 이렇게 하면 반드시 하나의 0 또는 1을 남길 수 있습니다.

C. 숫자 놀이

- 마지막으로 남은 0 또는 1과 N을 골라서 N또는 $N-1$을 만들면 이것이 최적입니다.

D. 축지법

`graph_traversal, set`

출제진 의도 – **Medium**

- 제출 125번, 정답 13팀 (정답률 10.400%)
- 처음 푼 팀: **goraani never stop**서강대학교, 40분
- 출제자: 전우제kiwiyou

D. 축지법

- 출발점과 도착점이 다른 연결 요소라면 답은 0분 혹은 1분입니다.
- 출발점과 도착점이 같은 연결 요소이고 유일하지 않다면,
  - 다른 연결 요소로 이동했다 돌아오면 2분에 이동할 수 있습니다.
  - 가능하다면 도로만을 사용하여 0분 혹은 1분에 이동할 수 있습니다.
- 출발점과 도착점이 같은 연결 요소이고 유일하다면 직접 구해야 합니다.

D. 축지법

- 같은 연결 요소인지 판단하기에 정점의 수가 너무 많습니다.
- 도로가 하나라도 연결되어 있는 지점의 개수는 최대 $2M$개라는 점을 이용합니다.
- 최대 $2M$개의 지점들에 맵을 이용하여 1부터 번호를 새로 매깁니다.
- 번호가 새로 매겨지지 않은 지점이 질문에 등장하면 답은 반드시 0 또는 1입니다.

D. 축지법

- 이제 최대 $2M$개의 정점으로 이루어진 그래프에 대한 문제로 바뀌었습니다.
- 번호가 매겨진 각 지점들로부터 그래프 탐색을 시행합니다.
- $\mathcal{O}\left(M^2\right)$ 시간에 각 지점 사이의 거리 $d(s, e)$를 구할 수 있습니다.
- 그래프 탐색을 통해 연결 요소의 개수 또한 구할 수 있습니다.

D. 축지법

- 연결 요소의 개수가 2 이상이라면 답은 $\min\left(d\left(s,e\right),2\right)$ 입니다.
- 연결 요소의 개수가 1 이라면 답은 $d(s,e)$ 입니다.
- 전체 시간복잡도 $\mathcal{O}\left(M^2+Q\log M\right)$ 에 문제를 해결할 수 있습니다.

E. n, 3n, 5n

`number_theory, priority_queue`

출제진 의도 – **Hard**

- 제출 64번, 정답 3팀 (정답률 4.668%)
- 처음 푼 팀: **탈PS는지능순**[연세대학교], 93분
- 출제자: 곽우석[bubbler]

E. n, 3n, 5n

- 3과 5가 모두 소수이므로, 다음과 같은 무한 격자를 상상할 수 있습니다.
- 맨 왼쪽 아래에 3의 배수도 아니고 5의 배수도 아닌 수를 하나 두고, 오른쪽으로 갈 때는 3을 곱하고, 위로 갈 때는 5를 곱합니다.

$\cdots$	$\cdots$	$\cdots$	$\cdots$	
1×5^3	$1 \times 3^1 \times 5^3$	$1 \times 3^2 \times 5^3$	$1 \times 3^3 \times 5^3$	$\cdots$
1×5^2	$1 \times 3^1 \times 5^2$	$1 \times 3^2 \times 5^2$	$1 \times 3^3 \times 5^2$	$\cdots$
1×5^1	$1 \times 3^1 \times 5^1$	$1 \times 3^2 \times 5^1$	$1 \times 3^3 \times 5^1$	$\cdots$
1	1×3^1	1×3^2	1×3^3	$\cdots$

E. n, 3n, 5n

- 맨 왼쪽 아래의 수에 따라 무한히 많은 격자를 만들 수 있고, 모든 양의 정수는 그 격자들 중 정확히 하나의 격자에 포함되는 것을 알 수 있습니다.
- 구체적으로, $n = 3^x 5^y z \, (\gcd(z, 15) = 1)$이면 n은 맨 왼쪽 아래의 수가 z인 격자의 (x, y)의 위치에 나타나게 됩니다.
- 임의의 n에 대해 $n, 3n, 5n$은 모두 같은 격자에 나타나므로, 하나의 격자 상에서 수열에 나타나는 수들을 구할 때 다른 격자의 수들은 관여하지 않게 됩니다.
- 그리고 하나의 격자에서 수열에 등장하는 수를 모두 찾으면 모든 격자의 같은 위치의 수들이 문제의 수열을 이루게 됩니다.

E. n, 3n, 5n

- 그럼 이제 아래의 격자에 대해서 문제를 풀어 봅시다.
- 문제의 조건은 임의의 (x, y)에 대해 $(x, y), (x+1, y), (x, y+1)$에 있는 3개의 수들 중 정확히 하나가 수열에 포함됨을 의미합니다.

$\cdots$	$\cdots$	$\cdots$	$\cdots$	
1×5^3	$1 \times 3^1 \times 5^3$	$1 \times 3^2 \times 5^3$	$1 \times 3^3 \times 5^3$	$\cdots$
1×5^2	$1 \times 3^1 \times 5^2$	$1 \times 3^2 \times 5^2$	$1 \times 3^3 \times 5^2$	$\cdots$
1×5^1	$1 \times 3^1 \times 5^1$	$1 \times 3^2 \times 5^1$	$1 \times 3^3 \times 5^1$	$\cdots$
1	1×3^1	1×3^2	1×3^3	$\cdots$

E. n, 3n, 5n

- 예제에서 맨 왼쪽 아래의 1이 수열에 포함됨을 알 수 있습니다.
- 그러면 3과 5는 수열에서 제외됩니다.

$\cdots$	$\cdots$	$\cdots$	$\cdots$	
1×5^3	$1 \times 3^1 \times 5^3$	$1 \times 3^2 \times 5^3$	$1 \times 3^3 \times 5^3$	$\cdots$
1×5^2	$1 \times 3^1 \times 5^2$	$1 \times 3^2 \times 5^2$	$1 \times 3^3 \times 5^2$	$\cdots$
1×5^1	$1 \times 3^1 \times 5^1$	$1 \times 3^2 \times 5^1$	$1 \times 3^3 \times 5^1$	$\cdots$
1	1×3^1	1×3^2	1×3^3	$\cdots$

E. n, 3n, 5n

- 남은 수 중 가장 작은 수는 9이므로, 9가 수열에 포함된다고 가정해 봅시다.
- 그러면 $(1,0)$의 삼각형 조건에 의해 $(1,1)$이, $(2,0)$의 조건에 의해 $(2,1)$이 수열에서 제외됩니다.

$\cdots$	$\cdots$	$\cdots$	$\cdots$	
1×5^3	$1\times 3^1\times 5^3$	$1\times 3^2\times 5^3$	$1\times 3^3\times 5^3$	$\cdots$
1×5^2	$1\times 3^1\times 5^2$	$1\times 3^2\times 5^2$	$1\times 3^3\times 5^2$	$\cdots$
1×5^1	$1\times 3^1\times 5^1$	$1\times 3^2\times 5^1$	$1\times 3^3\times 5^1$	$\cdots$
1	1×3^1	1×3^2	1×3^3	$\cdots$

E. n, 3n, 5n

- 이제 $(0, 1)$의 조건에 의해 25가, $(1, 1)$의 조건에 의해 75가 수열에 포함되어야 합니다.
- 그러나 이는 모순입니다. 따라서 9는 수열에 등장할 수 없습니다.

$\cdots$	$\cdots$	$\cdots$	$\cdots$	
1×5^3	$1 \times 3^1 \times 5^3$	$1 \times 3^2 \times 5^3$	$1 \times 3^3 \times 5^3$	$\cdots$
1×5^2	$1 \times 3^1 \times 5^2$	$1 \times 3^2 \times 5^2$	$1 \times 3^3 \times 5^2$	$\cdots$
1×5^1	$1 \times 3^1 \times 5^1$	$1 \times 3^2 \times 5^1$	$1 \times 3^3 \times 5^1$	$\cdots$
1	1×3^1	1×3^2	1×3^3	$\cdots$

E. n, 3n, 5n

- $(1, 0)$ 조건에 의해 15가 수열에 등장해야 합니다.
- 조건을 계속 적용하여 패턴을 조금 그려나가 보면 다음과 같은 모양이 나오게 됩니다.

$\cdots$	$\cdots$	$\cdots$	$\cdots$	
1×5^3	$1 \times 3^1 \times 5^3$	$1 \times 3^2 \times 5^3$	$1 \times 3^3 \times 5^3$	$\cdots$
1×5^2	$1 \times 3^1 \times 5^2$	$1 \times 3^2 \times 5^2$	$1 \times 3^3 \times 5^2$	$\cdots$
1×5^1	$1 \times 3^1 \times 5^1$	$1 \times 3^2 \times 5^1$	$1 \times 3^3 \times 5^1$	$\cdots$
1	1×3^1	1×3^2	1×3^3	$\cdots$

E. n, 3n, 5n

- 이렇게 해서 수열에 포함되는 수들은 $x \equiv y \pmod 3$ 인 (x, y) 에 있는 수들이라는 결론을 얻습니다.
- 이제 그러한 수들을 모두 구하기 위해서는 먼저 min heap에 15와 서로소인 수들을 충분히 많이 넣어놓고 x를 꺼낼 때마다 $15x, 27x, 125x$를 힙에 넣으면 됩니다.
- 이때 중복 방지를 위해 $125x$는 x가 3의 배수가 아닐 때만, $27x$는 x가 5의 배수가 아닐 때만 넣는 방법 등을 사용할 수 있습니다.
- 시간 제한이 넉넉하기 때문에 트리셋 등을 사용해도 됩니다.

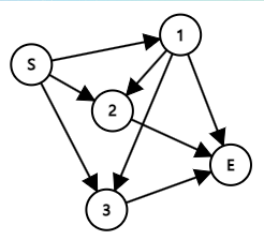
F. 사탕 배달

`flow`, `dp`

출제진 의도 – **Challenging**

- 제출 0번, 정답 0팀 (정답률 0.000%)
- 처음 푼 팀: —[대학교], 0분
- 출제자: 정치훈[ANZ1217]

F. 사탕 배달

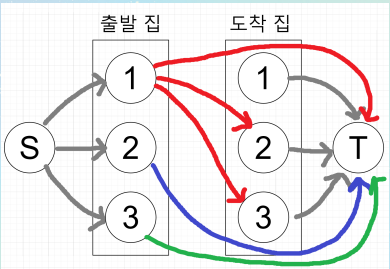


- 일단 집이 최대 K개이기 때문에, 500×500크기의 그래프를 만들 필요는 없습니다.
- 시작지점과 도착지점, 그리고 모든 집 사이의 거리를 DP등으로 구해 정점이 $K+2$개인 그래프로 변환합시다.

F. 사탕 배달

- 배달 순서는 문제를 푸는데 불필요합니다. 배달 순서와 상관 없이, 아무튼 모든 집이 한 번씩 방문되면 됩니다. 그러니 각각의 집을 독립적으로 처리해 봅시다.
- 어떤 집에 이미 도착했다고 가정한다면, 다음에 안즈가 할 행동은 둘 중 하나입니다.
  - 1. 이동할 수 있는 다른 집으로 이동한다.
  - 2. (N, M)으로 이동한다.

F. 사탕 배달



- 집(정점)을 분리해서 그래프로 모델링 해보면 위 같은 모양이 됩니다.
- 어디서 많이 본 그래프 형태죠? 유량을 흘려봅시다.

F. 사탕 배달

- 각 간선은 유량 (최대 1번)과 비용 (정점 사이의 거리)가 존재합니다.
- 모든 집에 한 번씩 방문해야 된다는 조건은 최대 유량(K)이 흐르면 자연스럽게 만족합니다. 따라서 이 그래프의 간선에 적절하게 cost를 설정하고, MCMF를 계산하면 됩니다.
- 시간복잡도는 유량이 최대 $O(K)$, 정점이 $O(K)$개인 그래프에서의 MCMF이므로 $O(K^4)$입니다.

G. 지워진 ETT

`dp, combinatorics`

출제진 의도 – **Medium**

- 제출 18번, 정답 5팀 (정답률 27.778%)
- 처음 푼 팀: **Redshift**[서강대학교], 83분
- 출제자: 김도훈[dohoon]

G. 지워진 ETT

- 카탈란수, 괄호 문자열과 관련이 있습니다.
- 한 정점에 들어오는 순간, 나오는 순간이 괄호를 열고, 닫는 것에 대응되기 때문입니다.
- 따라서 카탈란수 점화식과 비슷하게 $O(n^3)$의 시간복잡도로 계산할 수 있습니다.
- 그 외에 까다로운 점은, 부여되지 않은 순서들은 무시해 두었다가 점화식을 모두 계산한 뒤에 팩토리얼을 취하여 곱해야 한다는 것 정도가 있습니다.

H. 잊음을 논함

`tree`, `splay_tree`, `link_cut_tree`

출제진 의도 – **Challenging**

- 제출 5번, 정답 0팀 (정답률 0.000%)
- 처음 푼 팀: -[대학교], 0분
- 출제자: 노현제[hjroh0315]

H. 잊음을 논함

- 시간 복잡도와 공간 복잡도가 서로 다른 여러 풀이가 있습니다.
- 본 에디토리얼에서는 가장 단순한 것부터 순차적으로 설명합니다.

H. 잊음을 논함

- 시간 복잡도 $\mathcal{O}(Q^2)$, 공간 복잡도 $\mathcal{O}(N+Q)$의 가장 단순한 풀이가 있습니다.
- 지금까지 들어온 업데이트를 s 쿼리가 들어온 시간 순으로 저장합시다.
- 이 업데이트를 모두 반영한 순열을 출력 전에 재계산합니다.
- 이 재계산을 업데이트 시점에 수행하면 업데이트당 $\mathcal{O}(Q)$, 출력당 $\mathcal{O}(1)$이 됩니다.
- 이 재계산을 출력 직전 시점에 수행하면 업데이트당 $\mathcal{O}(1)$, 출력당 $\mathcal{O}(Q)$가 됩니다.

H. 잊음을 논함

- 두 시간 복잡도 간의 tradeoff가 가능해 보입니다.
- 상수 B를 선택한 뒤, 업데이트를 B개씩 묶어 처리합시다.
- 업데이트 시점에는 같은 버킷의 업데이트를 반영한 순열을 재계산합니다.
- 출력 시점에는 뒤쪽의 순열부터 각각 합성하여 얻은 순열에서 값을 출력합니다.
- 이때 시간 복잡도는 업데이트당 $\mathcal{O}(B)$, 출력당 $\mathcal{O}(Q/B)$가 됩니다.
- $B = \sqrt{Q}$를 선택하면 각 쿼리당 $\mathcal{O}(\sqrt{Q})$, 총 $\mathcal{O}(Q\sqrt{Q})$의 시간 복잡도를 얻습니다.
- 재계산을 위해 배열을 롤백하는 경우 $\mathcal{O}(N\sqrt{Q})$의 공간 복잡도를 얻습니다.
- 재계산을 위해 해시 테이블을 초기화하는 경우 $\mathcal{O}(N+Q)$의 공간 복잡도를 얻습니다.

H. 잊음을 논함

- 접근을 바꾸지 않고 시간 복잡도의 비약적인 개선을 얻기는 어렵습니다.

H. 잊음을 논함

- 중요한 관찰 한 가지로 완전히 새로운 풀이를 얻을 수 있습니다.
  - 두 순열 p와 q가 있을 때, p에서 두 자리가 바뀌면 $p \circ q$에서도 두 자리가 바뀝니다.
  - 마찬가지로, q에서 두 자리가 바뀌면 $p \circ q$에서도 두 자리가 바뀝니다.
- 각 노드가 구간 내의 모든 업데이트를 반영한 순열을 저장하는 세그먼트 트리를 관리합시다.
- 한 리프 노드의 순열에서 두 자리가 바뀌면, 총 $\mathcal{O}(\log Q)$ 개의 노드에서도 동시에 두 자리가 바뀝니다.
- 리프에서 루트로 올라가면서 바꾸어야 할 자리를 구하여 바꾸어 줄 수 있습니다.

H. 잊음을 논함

- 순열을 저장할 때는 $i \to i$를 모두 제외하고 저장합니다.
- 각 층에서 최대 $2Q$개의 자리가 바뀌어 있으므로, 공간 복잡도는 $\mathcal{O}(Q \log Q)$가 됩니다.
- 각 노드에서 관리하는 자료 구조의 연산당 시간 복잡도가 $\mathcal{O}(T)$라면, 총 시간 복잡도는 $\mathcal{O}(TQ \log Q)$입니다.
- 자료 구조로 BBST를 선택하면 $\mathcal{O}(Q \log^2 Q)$의 시간 복잡도를 얻습니다.
- 자료 구조로 해시 테이블을 선택하면 평균적으로 $\mathcal{O}(Q \log Q)$의 시간 복잡도를 얻습니다.

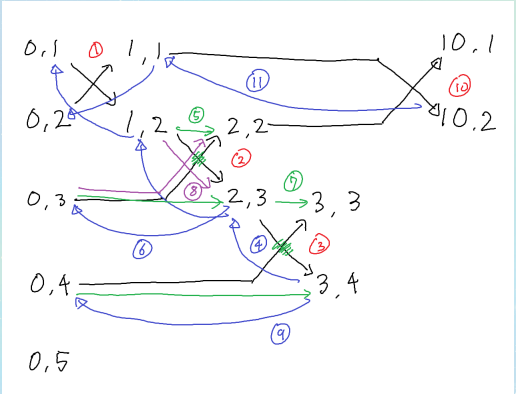
H. 잊음을 논함

- $\mathcal{O}(Q \log Q)$의 시간 복잡도와 $\mathcal{O}(N+Q)$의 공간 복잡도를 얻을 수 있습니다.
- 다음과 같은 시선에서 s 질의를 바라봅시다.
  - i번째 값이 마지막으로 변경된 시점이 $last_i$라고 합시다.
  - 초기에 $last_i$는 모든 i에 대해 0입니다.
  - "시각 t에 i번째 값이 업데이트되었다"를 "사건 (t, i)"로 표기합시다.
  - 초기에 N개의 사건 $(0,1), (0,2), \cdots, (0,N)$이 있습니다.
  - s i j 질의가 들어오면 사건 (t, i)와 사건 (t, j)가 발생합니다.
  - 또한 두 개의 사건은 각각 사건 $(last_j, j)$, 사건 $(last_i, i)$와 연결됩니다.
  - 그후 $last_i$, $last_j$가 모두 t로 업데이트됩니다.

H. 잊음을 논함

- 이제 사건 하나가 정점 하나에 해당하는 그래프를 생각합시다.
- 이 그래프는 각 정점이 하나의 루트 $(0, x)$를 가지는 루트 있는 포레스트입니다.
- 그러므로 각각의 질의를 링크-컷 트리의 동작에 대응시킬 수 있습니다.
  - s 질의는 두 개의 정점을 만들고 두 번의 link 동작을 합니다.
  - f 질의와 r 질의는 두 번의 cut 동작과 두 번의 link 동작을 합니다.
  - q 질의는 정점 $(last_i, i)$의 루트를 찾아 출력합니다.
- 이렇게 하면 쿼리당 amortized $\mathcal{O}(\log Q)$의 시간 복잡도를 얻습니다.
- 공간 복잡도는 사건의 수에 비례하므로 $\mathcal{O}(N + Q)$입니다.

H. 잊음을 논함



- 예제 입력을 선술한 시선에서 시각화하면 위와 같습니다.

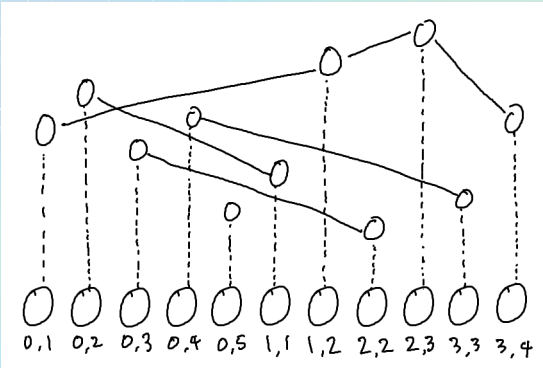
H. 잊음을 논함

- 링크-컷 트리를 사용해야만 풀리는 문제라고 하기에는 어딘가 아쉽습니다.
- 사실 링크-컷 트리를 사용하지 않아도 해결할 수 있습니다.
- 루트있는 포레스트 안의 각 트리는 하나의 경로를 이룹니다.
- 경로 위의 사건 (t, i)를 모두 나열하면, 경로 위의 t는 단조증가합니다.
- 따라서 Split과 Join이 가능한 BBST를 사용해도 해결할 수 있습니다.

H. 잊음을 논함

- 예시로, 각각의 질의를 스플레이 트리의 동작에 대응시키면 다음과 같습니다.
  - `s` 질의는 두 노드를 splay하고 새로운 두 노드를 오른쪽 노드로 붙입니다.
  - `f` 질의와 `r` 질의는 두 노드를 splay하고 왼쪽 서브트리를 교환합니다.
  - `q` 질의는 트리에서 최솟값에 해당하는 노드를 찾습니다.
- 이렇게 하면 쿼리당 amortized $\mathcal{O}(\log Q)$의 시간 복잡도를 얻습니다.
- 공간 복잡도는 그대로 $\mathcal{O}(N+Q)$입니다.

H. 잊음을 논함



- 예제 입력 (3번째 질의 시점) 을 스플레이 트리로 표현한 것을 시각화하면 위와 같습니다.

H. 잊음을 논함

- 출제자 코드의 실행 시간과 사용 메모리는 다음과 같습니다.
  - 스플레이 트리 1396ms, 127576KB
  - 트립 2748ms, 158904KB
- 상수가 크면 (예를 들어, `std::set`을 사용할 때) 같은 시간 복잡도를 가정하더라도 통과하지 못할 가능성이 존재합니다.

I. 시험 주행

`geometry`, `case_work`

출제진 의도 – **Easy**

- 제출 55번, 정답 11팀 (정답률 20.000%)
- 처음 푼 팀: ALL_TIME_LEGEND[서강대학교], 25분
- 출제자: 김영현[kipa00]

I. 시험 주행

- 주어지는 입력의 순서를 바꾸어 항상 P_1에 가장 빠른 차량이 있게 할 수 있습니다.
- 한 차량만 있다면, 각 차량이 P_1, P_2, P_3에 최초로 도달하는 시간을 간단한 식을 통해 계산할 수 있습니다.
- 모든 $i = 1, 2, 3$, 모든 $C = \text{slow}, \text{fast}$에 대해,

$$T_{i,C} = \begin{cases} 0 & P_i\text{가 차량 } C\text{의 시작 지점인 경우} \\ T_{(i-2) \bmod 3+1,C} + \dfrac{\text{dist}(P_{(i-2) \bmod 3+1}, P_i)}{v_C} & \text{otherwise} \end{cases}$$

입니다.

I. 시험 주행

- 빠른 차량이 느린 차량을 추월하지 못하는 유일한 공간인 1-차로는 정확히 한 군데에만 존재합니다.
- 1-차로가 어디 있는지에 따라 경우를 나누어 봅시다.
- 먼저 P_1–P_2가 1-차로라면, 빠른 차량이 P_1에 먼저 진입합니다.
- 1-차로에 빠른 차량이 먼저 진입하는 경우, 빠른 차량이 느린 차량을 이 차로에서 추월하지 않습니다.

I. 시험 주행

- 다음으로 P_2–P_3가 1차로인 경우를 생각합시다.
- 느린 차량이 처음에 P_2에 있었다면, 느린 차량이 P_3로 빠져나가는 시각보다 빠른 차량이 P_3로 빠져나가는 시각이 더 빠르면 빠른 차량은 추월을 원하게 됩니다.
- 느린 차량이 처음에 P_3에 있었다면, 빠른 차량과 느린 차량이 이 차로 위에 동시에 존재할 수 없습니다.
- 따라서 조건은 느린 차량의 초기 위치가 P_2이고 $T_{3,\text{fast}} < T_{3,\text{slow}}$인 경우입니다.
- 이때 빠른 차량의 정답은 $T_{3,\text{slow}}$에 P_3–P_1을 이동하는 시간의 합입니다.

I. 시험 주행

- 마지막으로 P_3–P_1 이 1차로인 경우를 생각합시다.
- 차량 하나만 있었을 때 다음 상황이어야 이 도로에서 빠른 차량이 추월을 원할 수 있습니다:
  1. 느린 차량이 P_3에 진입
  2. 빠른 차량이 P_3에 진입
  3. 빠른 차량이 P_1으로 진출 (한 바퀴를 완전히 돎)
  4. 느린 차량이 P_1으로 진출
- 이를 수식으로 쓰면 $T_{3,\text{slow}} < T_{3,\text{fast}}$ 그리고 $T_{3,\text{fast}} + \text{dist}(P_3, P_1)/v_{\text{slow}} < T_{1,\text{slow}}$ 입니다. 빠른 차량의 답은 단순히 $T_{1,\text{slow}}$ 입니다.

I. 시험 주행

- 실수 오차 때문에 답이 현저히 차이 나는 경우에 대해 주의합시다.
- 실수 오차로 인한 판정으로 답이 현저히 차이 나는 경우는 추월 시점 앞뒤로 추월까지의 시간이 길 때입니다.
- 즉, 1-차로에 진입해야 하는 시점과 최대한 겹치는 경우입니다.
- 1-차로가 P_1–P_2인 경우는 고려하지 않고, P_2–P_3인 경우는 $T_{2,\text{slow}} = 0$이므로 걱정할 것이 없습니다.

I. 시험 주행

- 1-차로가 P_3–P_1 인 경우, P_3 의 진입 시점의 대소 관계를 잘못 비교하는 경우 답이 매우 크게 차이 날 수 있습니다.
- 느린 차량의 초기 위치가 P_3 라면, $T_{3,\text{slow}} = 0$ 이므로 걱정할 것이 없습니다.
- 느린 차량의 초기 위치가 P_2 라면,

$$T_{3,\text{slow}} = \frac{\text{dist}(P_2, P_3)}{v_{\text{slow}}} \overset{?}{<} \frac{\text{dist}(P_1, P_2) + \text{dist}(P_2, P_3)}{v_{\text{fast}}} = T_{3,\text{fast}}$$

인지 비교해야 합니다.

I. 시험 주행

- 이는 다음 변형을 통해 정확히 비교할 수 있습니다. dist^2는 정수 값입니다.

$$\begin{aligned}
\frac{\mathrm{dist}(P_2, P_3)}{v_{\mathrm{slow}}} &\overset{?}{<} \frac{\mathrm{dist}(P_1, P_2) + \mathrm{dist}(P_2, P_3)}{v_{\mathrm{fast}}} \\
\frac{v_{\mathrm{fast}}}{v_{\mathrm{slow}}} &\overset{?}{<} \frac{\mathrm{dist}(P_1, P_2) + \mathrm{dist}(P_2, P_3)}{\mathrm{dist}(P_2, P_3)} \\
\frac{v_{\mathrm{fast}}}{v_{\mathrm{slow}}} &\overset{?}{<} \frac{\mathrm{dist}(P_1, P_2)}{\mathrm{dist}(P_2, P_3)} + 1 \\
\frac{v_{\mathrm{fast}} - v_{\mathrm{slow}}}{v_{\mathrm{slow}}} &\overset{?}{<} \frac{\mathrm{dist}(P_1, P_2)}{\mathrm{dist}(P_2, P_3)} \\
\mathrm{dist}^2(P_2, P_3)\,(v_{\mathrm{fast}} - v_{\mathrm{slow}})^2 &\overset{?}{<} \mathrm{dist}^2(P_1, P_2) v_{\mathrm{slow}}^2
\end{aligned}$$

I. 시험 주행

- 다행히도, 이러한 실수 오차로 비교가 잘못될 만큼 제한이 크지 않기 때문에, `double`을 사용해도 큰 걱정 없이 해결할 수 있습니다 🤩🤩
  - 걱정이 된다면 세터를 믿고 그냥 `long double`을 사용합시다!
- 2024년에 `float`은 좀 너무한 거 같아서 이건 저격했습니다 😈😈😈

I. 시험 주행

- 여기서부터는 실수 오차의 하한 증명입니다.
- 좌표의 입력 제한을 L, 속도의 입력 제한을 V라 합시다.
- $l_1 = \mathrm{dist}(P_1, P_2)$, $l_2 = \mathrm{dist}(P_2, P_3)$라 할 때, $\frac{l_1+l_2}{v_{\mathrm{fast}}}$와 $\frac{l_2}{v_{\mathrm{slow}}}$의 값을 비교하면 됩니다.

I. 시험 주행

- 절대 오차를 계산합니다:

$$\begin{aligned}\frac{l_1+l_2}{v_{\text{fast}}}-\frac{l_2}{v_{\text{slow}}} &= \frac{v_{\text{slow}}l_1-(v_{\text{fast}}-v_{\text{slow}})l_2}{v_{\text{fast}}v_{\text{slow}}} \\ &= \frac{v_{\text{slow}}^2l_1^2-(v_{\text{fast}}-v_{\text{slow}})^2l_2^2}{v_{\text{fast}}v_{\text{slow}}(v_{\text{slow}}l_1+(v_{\text{fast}}-v_{\text{slow}})l_2)} \\ &= \frac{v_{\text{slow}}^2l_1^2-(v_{\text{fast}}-v_{\text{slow}})^2l_2^2}{v_{\text{fast}}v_{\text{slow}}^2(l_1-l_2)+v_{\text{fast}}^2v_{\text{slow}}l_2}\end{aligned}$$

- 빨간색 부분이 정수이므로, 두 값이 같지 않다면 분자의 하한은 1입니다.

I. 시험 주행

$$\frac{l_1 + l_2}{v_{\text{fast}}} - \frac{l_2}{v_{\text{slow}}} = \frac{v_{\text{slow}}^2 l_1^2 - (v_{\text{fast}} - v_{\text{slow}})^2 l_2^2}{v_{\text{fast}} v_{\text{slow}}^2 (l_1 - l_2) + v_{\text{fast}}^2 v_{\text{slow}} l_2}$$

- 분모의 l_2가 증가할수록 $v_{\text{fast}}^2 v_{\text{slow}}$의 계수가 커지는 만큼 $v_{\text{fast}} v_{\text{slow}}^2$의 계수가 작아지므로, l_2가 가장 클 때 분모가 가장 커집니다.
- 따라서 분모는 $l_1 = l_2 = 2\sqrt{2}L$, $v_{\text{fast}} = v_{\text{slow}} = V$일 때 $2\sqrt{2}LV^3$로 최대가 됩니다.
- 두 값이 같지 않으면, 오차의 하한은 $cL^{-1}V^{-3}$입니다.

I. 시험 주행

$$\left(\frac{l_1+l_2}{v_{\text{fast}}}-\frac{l_2}{v_{\text{slow}}}\right)\Big/\frac{l_2}{v_{\text{slow}}}=\frac{v_{\text{slow}}^2 l_1^2-(v_{\text{fast}}-v_{\text{slow}})^2 l_2^2}{v_{\text{fast}}v_{\text{slow}}(l_1-l_2)l_2+v_{\text{fast}}^2 l_2^2}$$

- 마찬가지로 두 값이 같지 않은 경우 상대 오차의 하한은 $cL^{-2}V^{-2}$ 입니다.
- 상대 오차의 하한이 machine precision보다 100 배 이상 차이가 나므로, 두 값이 같은 경우만 잘 비교해 주면 됩니다!

J. a11y

`bitset`

출제진 의도 – **Hard**

- 제출 43번, 정답 3팀 (정답률 6.977%)
- 처음 푼 팀: PS:Endgame[연세대학교], 25분
- 출제자: 박상수[molamola]

J. a11y

- 입력 문자열을 $A_{1 \dots N}$이라 합시다.
- 각 알파벳 c에 대해 아래와 같은 길이 N의 bitset을 만듭니다.
- $X_{c,i}$: $A_i = c$이면 1, 아니면 0

J. a11y

- 쿼리에서 첫 글자를 s, 사이 문자 수를 m, 마지막 글자를 e라 합시다.
- $X_{s,*}$를 $m+1$칸 shift한 것과 $X_{e,*}$에서 둘 다 1인 것의 개수를 구하면 됩니다.

J. a11y

- C++의 경우 std::bitset에서 지원하는 shift, and, count 연산을 사용하면 됩니다.

```
while(Q--) {
    char s, e;
    int m;
    cin >> s >> m >> e;
    cout << ((X[s] << (m + 1)) & X[e]).count() << "\n";
}
```

- 시간복잡도는 $\mathcal{O}(NQ/64)$ 입니다.

K. 완전 이진 트리와 쿼리

`dp_tree`, `math`

출제진 의도 – **Medium**

- 제출 8번, 정답 3팀 (정답률 37.500%)
- 처음 푼 팀: PS:Endgame 연세대학교, 95분
- 출제자: 정치훈ANZ1217

K. 완전 이진 트리와 쿼리

0. 공통

- 루트가 1번으로 바뀌지 않는다고 가정합시다.
- 어떤 v번 정점을 루트로 하는 서브트리에 속한 모든 정점의 번호의 합은, $O(\log N)$에 쉽게 구할 수 있습니다.
- 이 합을 앞으로 $DP(v)$라고 합시다.
- 또, 루트가 1번일 때 v의 부모, 부모의 부모, $\cdots$ 등을 모두 v의 조상이라고 합시다.

K. 완전 이진 트리와 쿼리

1. 출제자 정해

- 쿼리를 오프라인으로 처리해 봅시다.
- 트리의 높이가 $\log N$이므로, 우리는 $O(Q \log N)$ 개의 정점만을 이용하면 입력으로 들어오는 모든 정점을 연결할 수 있습니다.
- 그렇게 필요한 정점만을 연결한 새로운 트리를 만듭시다.

K. 완전 이진 트리와 쿼리

- 새로운 트리에 속하는 정점의 $DP(v)$를 미리 계산해 놓읍시다.
- rerooting technique를 이용하면 DFS 한 번으로 이 트리에 속하는 모든 정점이 각각 새로운 루트가 될 때에 대해 $DP(v)$ 값을 구할 수 있습니다.
- 정점이 $O(Q \log N)$개, 각 정점에 대해 DP값 계산이 $O(\log N)$이므로 총 시간복잡도는 $O(Q \log^2 N)$ 입니다.

K. 완전 이진 트리와 쿼리

2. 더 쉬운 별해

- 2 v 쿼리가 들어올 때마다, v가 현재 루트의 조상인지 확인합시다.
- 조상이 아니라면 답은 그냥 $DP(v)$ 입니다.

K. 완전 이진 트리와 쿼리

- 그렇지 않다면, v의 조상들에 대해 답을 각각 계산한 다음 더해야 합니다.
- v의 조상 p의 두 자식에 대해, v가 없는 쪽의 DP값을 더해 나가면 됩니다.
- 이 방법 역시 쿼리당 $O(\log N)$개의 정점을 확인하고, 각 정점당 $O(\log N)$번의 추가 계산을 하기 때문에 시간복잡도는 $O(Q \log^2 N)$ 입니다.

L. 허술한 보안 프로그램

`ad_hoc, constructive, bitmask`

출제진 의도 – **Medium**

- 제출 56번, 정답 10팀 (정답률 17.857%)
- 처음 푼 팀: 차돌박이 된장찌개[연세대학교, 서강대학교], 51분
- 출제자: 서종환[mujigae]

L. 허술한 보안 프로그램

- 숨겨진 순열을 $\{h_n\}$, 입력한 순열을 $\{p_n\}$, 쿼리의 응답으로 얻은 수열을 $\{r_n\}$이라 하겠습니다.
- $\{r_n\}$은 $\{h_n\}$와 $\{p_n\}$의 각 원소에 대한 bitwise OR 연산 결과입니다.
- $\{p_n\}$의 원소들이 가진 비트 중에서 값이 0인 비트들을 이용한다는 생각을 해볼 수 있습니다.
- p_i의 j번째 비트가 0이라면 h_i의 j번째 비트는 r_i의 j번째 비트와 같습니다.
- 이를 이용하면 비트 단위로 해를 구성할 수 있게 됩니다. 이제 주어진 횟수 내에 문제를 해결해 봅시다.

L. 허술한 보안 프로그램

- 각 비트에 대해 0을 가진 원소가 몇 개 있는지 관찰해 봅시다.
- 자연수 N에 대해 주어진 순열은 0부터 $N-1$까지의 값을 가지므로 각 비트에 대해 0을 가진 원소의 개수는 $\lceil \frac{N}{2} \rceil$개 이상입니다. 전체 길이의 절반보다 많은 개수의 0을 가지고 있습니다.
- 첫 쿼리로 아무 순열을 입력하고 두 번째 쿼리에서는 첫 쿼리에서 알아내지 못한 비트, 즉 값이 1인 비트가 있던 곳에 0인 비트가 오는 순열을 입력해주면 $\mathbf{2}$번만에 답을 찾을 수 있습니다.

L. 허술한 보안 프로그램

- 이것을 쉽게 구현하기 위해 첫 번째 쿼리에 $0, 1, \cdots, N-1$까지 순서대로 증가하는 순열을 입력합니다.
- 다음으로 모든 비트에 대해 0은 1로, 1은 0으로 반전시킵니다. 여기서 $2^k < N \leq 2^{k+1}$를 만족하는 음이 아닌 정수 k에 대해 $k+1$번째 비트부터는 볼 필요가 없으므로 무시하겠습니다. 반전을 통해 순열이 $2^{k+1}-1, 2^{k+1}-2, \cdots, 2^{k+1}-N$으로 바뀝니다. 이제는 더 이상 범위에 맞는 순열이 아니므로 순열로 만들어 주어야 합니다. 이것은 각 비트의 0 또는 1의 개수를 세어 맞춤으로써 해결할 수 있습니다.

L. 허술한 보안 프로그램

- k번째 비트의 1의 개수가 원래 순열보다 많다면 이는 $N-1$을 초과하는 수의 개수가 원래 순열보다 많은 1의 개수만큼 있다는 것을 의미합니다. 현재 수열은 내림차순으로 정렬되어 있으므로 앞에서부터 1을 0으로 만들면 $2^k-1, 2^k-2, \cdots, 2^k-(2^{k+1}-N)$, $2^{k+1}-(2^{k+1}-N+1), \cdots, 2^{k+1}-N$이 됩니다.
- 다음으로 $2^{k+1}-(2^{k+1}-N+1), \cdots, 2^{k+1}-N$까지는 정상적인 순열의 범위에 들어 있으므로 그대로 가져가고 남은 앞부분 $2^k-1, \cdots, 2^k-(2^{k+1}-N)$을 생각합니다. 이것 또한 내림차순으로 정렬되어 있으므로 초과하는 1의 개수를 앞에서부터 0으로 바꾸어 맞춰주면 뒷부분을 순열의 일부로 만들 수 있습니다.

L. 허술한 보안 프로그램

- 이렇게 k번째 비트부터 시작하여 0번째 비트까지 내려가면서 현재 1의 개수가 원래 순열이 가지고 있어야 하는 개수와 같아질 때까지 앞에서부터 1을 0으로 바꿔줍니다.
- 이렇게 만들어진 수열은 첫 번째 쿼리가 얻지 못한 모든 비트에 대한 정보를 얻을 수 있게 하는 새로운 순열이 됩니다. 이렇게 두 번의 쿼리로 문제를 해결할 수 있습니다.

L. 허술한 보안 프로그램

- ex) $N = 9$

0	0	0	0	0	0	0	0	1		1	1	1	1	1	1	1	1	0		0	0	0	0	0	0	0	1	0
0	0	0	0	1	1	1	1	0		1	1	1	1	0	0	0	0	1		0	1	1	1	0	0	0	0	1
0	0	1	1	0	0	1	1	0	⟶	1	1	0	0	1	1	0	0	1	⟶	0	1	0	0	1	1	0	0	1
0	1	0	1	0	1	0	1	0		1	0	1	0	1	0	1	0	1		0	0	1	0	1	0	1	0	1
0	1	2	3	4	5	6	7	8		15	14	13	12	11	10	9	8	7		0	6	5	4	3	2	1	8	7

M. 나무 키우기

`segtree`

출제진 의도 – **Hard**

- 제출 31번, 정답 3팀 (정답률 9.677%)
- 처음 푼 팀: PS:Endgame 연세대학교, 116분
- 출제자: 박상수 `molamola`

M. 나무 키우기

- $18N$번 안에는 $\min\ A \times 2 \geq \max A$ 가 됩니다.
- 이 시점부터는 곱해지는 수가 계속 사이클을 돌게 됩니다.
- 이를 이용하면 $X \geq 18N$ 이후로는 답을 쉽게 구할 수 있습니다.

M. 나무 키우기

- X가 작은 경우는 직접 시뮬레이션해야 합니다.
- k번째 수 구하기, 삽입, 삭제가 되는 세그먼트 트리를 만들면 됩니다.

M. 나무 키우기

- 가장 작은 수가 $150,000$를 넘는 순간 시뮬레이션을 멈춥니다.
- 시간복잡도는 구현에 따라 $\mathcal{O}(M\log^2 M)$ 혹은 $\mathcal{O}(M\log M)$ 입니다.
- 이 때 M은 $\max A$와 N 중 큰 값입니다.